Executable Specifications

Foundations of Software Engineering
http://msrweb/foundations
April 16, 2002

The group

Yuri Gurevich

Wolfram Schulte

Michael Barnett

Wolfgang Grieskamp

Nikolai Tillmann

Jan. 1999

Jan. 2000

Jan. 2001

Jan. 2002

Marcus Veanes

Lev Nachmanson

Andreas Blass

Partners

- BizTalk
- Hailstorm
- Indigo
- ITE (formerly TMT)
- Netconfig
- Palladium (formerly Trusted Windows)
- ...

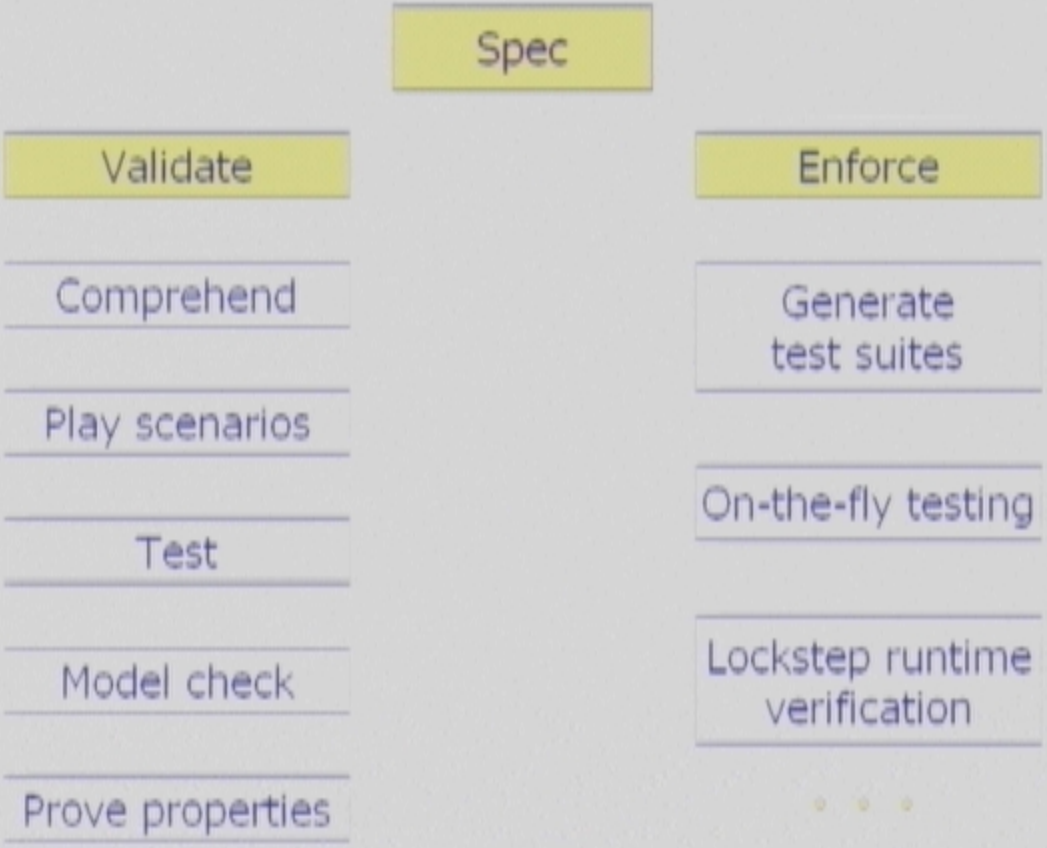
Problems we solve

- Poor specifications
  - Unenforceable
  - Poorly documented
- Growing spec. vs implementation gap
- High cost of design bugs
- Problems with testing
  - starts late in the process
  - often fails to test the intended functionality
  - is insufficiently automated

Our approach

Rigorous executable comprehensible spec

- Validate the design
  - Comprehend
  - Play scenarios
  - Test, model-check
  - Prove properties
- Enforce the spec
  - Generate test suites algorithmically
  - Use conformance execution


Spec
Validate
Enforce
Comprehend
Play scenarios
Test
Model check
Prove properties
Generate test suites
On-the-fly testing
Lockstep runtime verification

Conventional wisdom

- Specification should be declarative. Exec. spec is a contradiction in terms.
  - If you can execute the spec then why would you bother to implement it?
  - If you can execute the spec then it got to be full of irrelevant details, too specific, too low level.
- In fact, exec specs make sense

Example: topological sorting

- Given an acyclic digraph G = (V,E), arrange the vertices into a sequence S where each edge (u,v) leads forward.
- Observe: there is a vertex without incoming edges, and the remaining digraph is acyclic.
- Topsort: Use the observation repeatedly to build the desired sequence S.

Topsort

- A Modula-2 implementation by Niklaus Wirth
- An AsmL spec (a more OO version)

topsort_mod.txt - Notepad

File Edit Format View Help

```
(* Read a sequence of relations defining a directed, finite
   graph.  Then establish whether or not a partial ordering
   is defined.  If so, print the element in a sequence showing
   the partial ordering. (Topological sorting)  *)

MODULE topsort;

FROM InOut   IMPORT WriteLn, WriteCard, ReadCard, WriteString;
FROM Storage IMPORT ALLOCATE;

TYPE lref = POINTER TO leader;

     tref = POINTER TO trailer;

     leader = RECORD
                key: CARDINAL;
                count: CARDINAL;
                trail: tref;
                next: lref;
              END;

     trailer = RECORD
                 id: lref;
                 next: tref
               END;

VAR head,tail,p,q: lref;
```

Topological Sorting

```
class Vertex
```

This is an abstract class at this point - no fields or methods. We don't know what vertices are.

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []

topsort()
  step until fixpoint
    let X = V difference asSet(S)
    if X <> {} then
      S := S + [(any v | v in X where not exists u in X where E(u,v))]
```

In order to print vertices, we endow Vertex with an integer field and a printing method.

```
class Vertex...
  key as Integer
  asString() as String
    return "v" + asString(key)
```

We provide a particular input for the demo.

```
initializeDigraph()
  let N = [new Vertex(i) | i in [0..9]]
  V := asSet(N)
  E := {(N(6),N(0)),(N(2),N(0)),(N(8),N(2)), (N(8),N(1)), (N(8),N(2)),
        (N(5),N(1)),(N(1),N(0)),(N(9),N(1)),(N(9),N(4)),(N(4),N(0)),
        (N(7),N(3))/,(N(2),N(7))
}
```

This is the graphical form of the same.

Topological Sorting

```
class Vertex
```

This is an abstract class at this point - no fields or methods. We don't know what vertices are.

```
var V as Set of Vertex
var E as Set of (Vertex,Vertex)
var S as Seq of Vertex = []

topsort()
  step until fixpoint
    let X = V difference asSet(S)
    if X <> {} then
      S := S + [(any v | v in X where not exists u in X where E(u,v))]
```

In order to print vertices, we endow Vertex with an integer field and a printing method.

```
class Vertex...
  key as Integer
  asString() as String
    return "v" + asString(key)
```

We provide a particular input for the demo.

```
initialiseDigraph()
  let N = [new Vertex(i) | i in [0..9]]
  V := asSet(N)
  E := {(N(6),N(0)),(N(2),N(0)),(N(8),N(2)), (N(5),N(1)), (N(5),N(2)),
        (N(7),N(7)),(N(7),N(5)),(N(9),N(7)),(N(9),N(4)),(N(4),N(5)),
```

```
class Vertex...
  key as Integer
  asString() as String
    return "v" + asString(key)
```

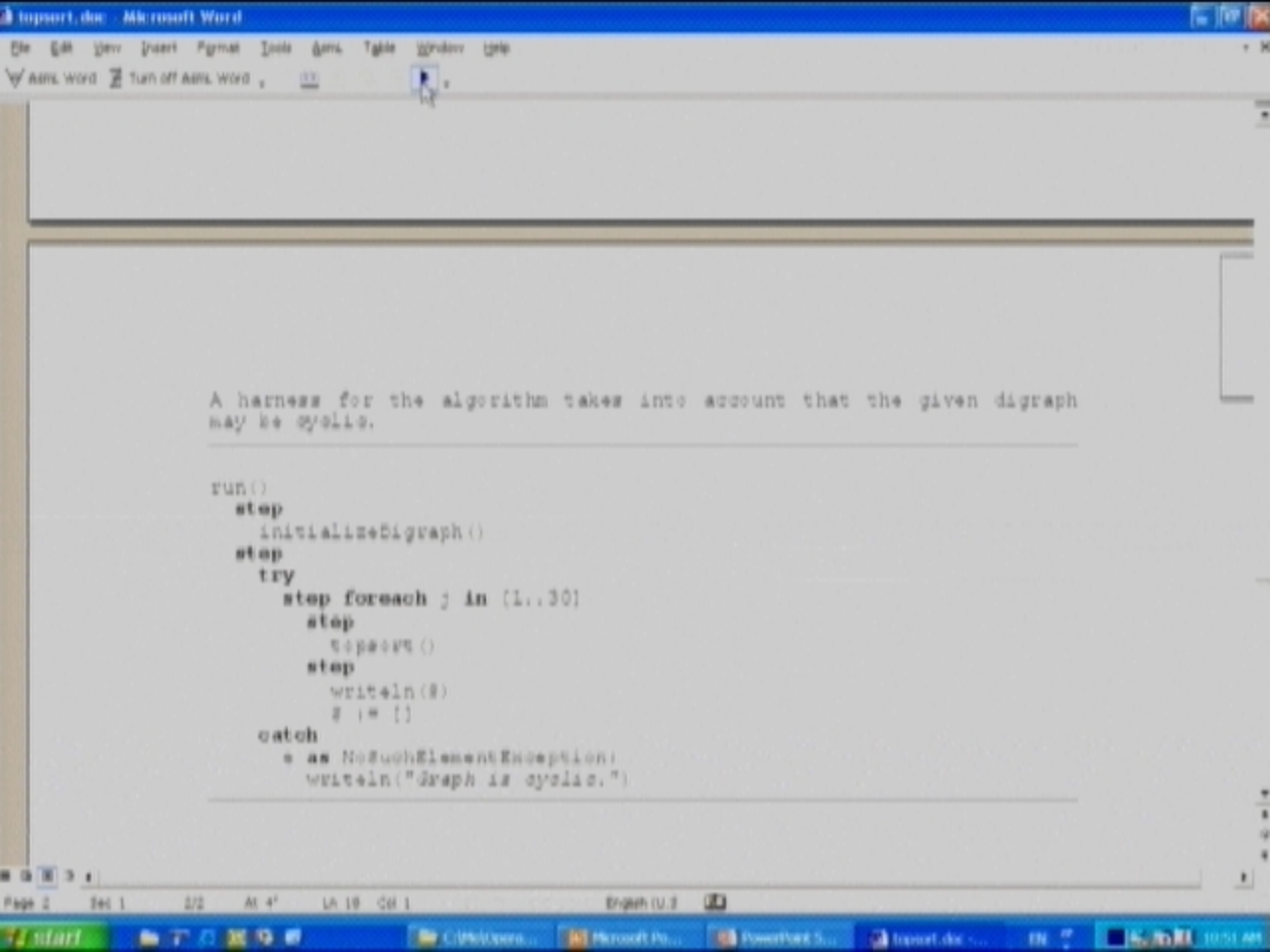
We provide a particular input for the demo:

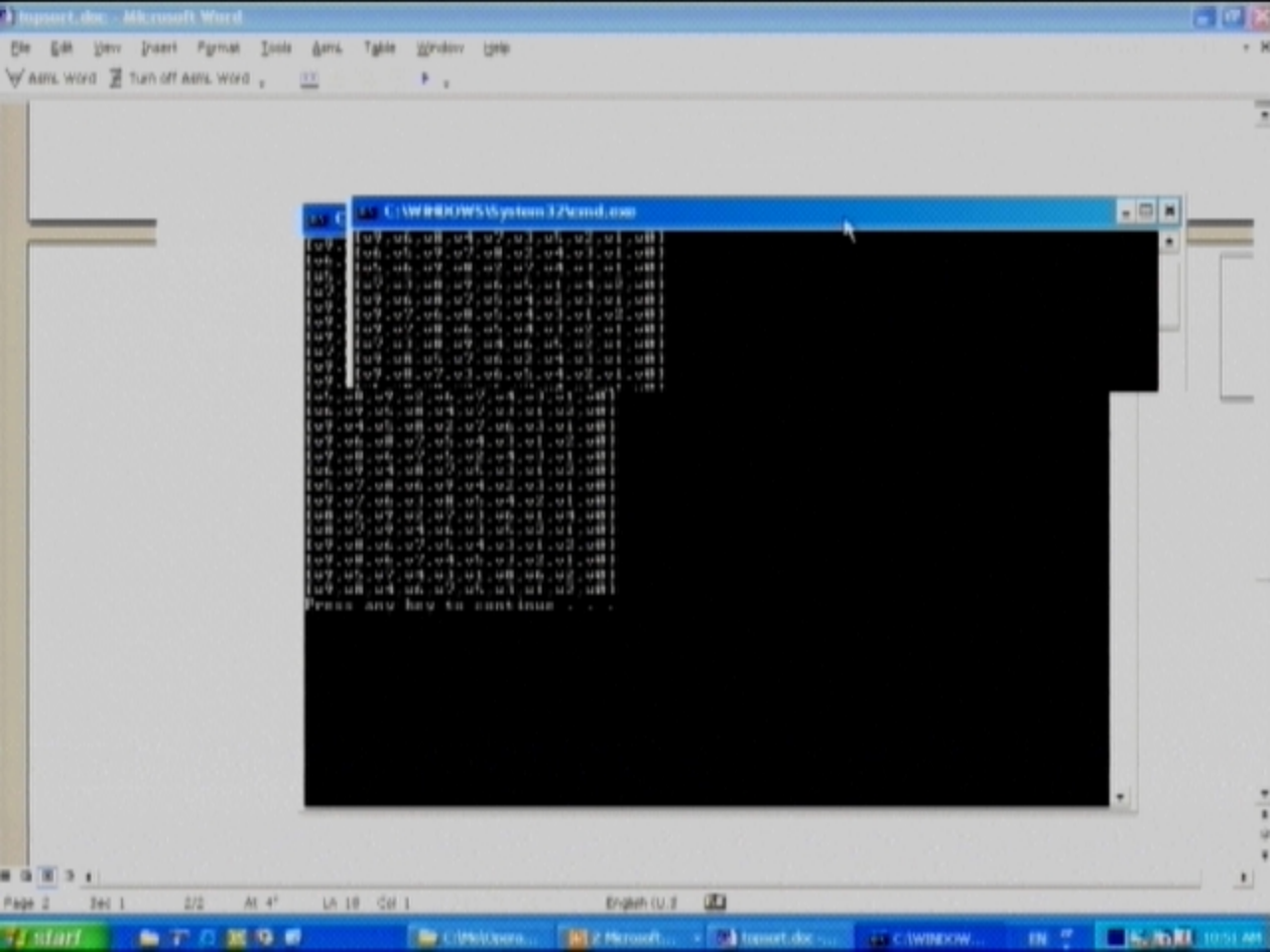
```
initialiseDigraph()
  let N = [new Vertex(i) | i in [0..9]]
  V := asSet(N)
  E := {(N(6),N(0)),(N(2),N(0)),(N(6),N(2)), (N(6),N(1)), (N(6),N(2)),
        (N(3),N(1)),(N(1),N(0)),(N(9),N(1)),(N(9),N(4)),(N(4),N(0)),
        (N(7),N(3))//,(N(0),N(7))
)
```

This is the graphical form of the same.

A harness for the algorithm takes into account that the given digraph may be cyclic.

```
run()
  step
    initialiseDigraph()
  step
    try
      step foreach j in [1..30]
        step
          toposort()
        step
          writeln(0)
          0 := []
    catch
      e as NoSuchElementException!
        writeln("Graph is cyclic.")
```


topsort.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
Add-ins Word Turn off Add-in Word
A harness for the algorithm takes into account that the given digraph may be cyclic.
run()
step
initializeDigraph()
step
try
step foreach j in [1..30]
step
topsort()
step
writeln(#)
# := []
catch
e as NoSuchElementException:
writeln("Graph is cyclic.")
Page 2 Sec 1 2/2 At 4" Ln 10 Col 1 English (U.S
start
EN
10:51 AM


C:\WINDOWS\system32\cmd.exe
Press any key to continue . . .

This is the graphical form of

C:\WINDOWS\System32\cmd.exe

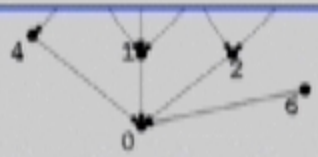
Press any key to continue . . .

```
  key as Integer
  asString() as String
    return "v" + asString(key)

We provide a particular input for the demo.

initializeDigraph()
  let H = [new Vertex(i) | i in [0..9]]
  V := asSet(H)
```

Axiom Compiler



```
  key as Integer
  asString() as String
    return "v" + asString(key)

We provide a particular input for the demo.

initializeDigraph()
  let N = [new Vertex(i) | i in [0..9]]
  V := asSet(N)
```

Aarst Compiler

Output | Errors

```
Microsoft (R) Aarts v1.0 Compiler v1.2a (2002/01)
Copyright (C) Microsoft Corporation. All rights reserved.
processing C:\Program Files\AIE\Aarts\include\runtime\prelude.aarts
processing C:\Program Files\AIE\Aarts\include\runtime\default.aarts
processing C:\Me\Opera\Talks\Aarts\Import\import.doc
Microsoft (R) Aarts v1.0 Word Extractor
Copyright (C) Microsoft Corporation. All rights reserved.
```

File Edit View Insert Format Tools Aami Table Window Help

Aami Word | Turn off Aami Word

```
  key as Integer
  asString() as String
    return "v" + asString(key)

We provide a particular input for the demo.

initializeDigraph()
  let N = [new Vertex(i) | i in [0..9]]
  V := asSet(N)
```

Aami Compiler

Output | Errors

```
Microsoft (R) Aami v1.0 Compiler v1.2a (2002/01)
Copyright (C) Microsoft Corporation. All rights reserved.
processing C:\Program Files\PSE\Aami\include\runtime\prelude.aami
processing C:\Program Files\PSE\Aami\include\runtime\default.aami
processing C:\Me\Opera\Talks\DAMSB\report\report.doc
Microsoft (R) Aami v1.0 Word Extractor
Copyright (C) Microsoft Corporation. All rights reserved.
connecting to Word...
extracting from report.doc .....
Warnings:
 unrepresentable formatting
```

Page 1 Sec 1 1/2 At 4.8" Ln 27 Col 20 German (Ge

Key as Integer
asString() as String
return "v" + asString(key)
We provide a particular input for the demo.
initializeDigraph()
let N = [new VertexN(i) | i in [0..9]]
Asml Compiler
Output
Errors
validating parse tree
analyzing declarations
analyzing definitions
transforming
reducing
generating C++ code
generating object code
calling c++ compiler
tassrt.obj
4
1
2
6
0

```
  step until fixpoint
    let X = V difference asSet(H)
    if X <> {} then
      H := H - {any V | V in X where not exists u in X where E(u,v)}
```

In order to print vertices, we endow Vertex with an integer field and a printing method.

```
class Vertex...
  key as Integer
  asString() as String
    return "v" + asString(key)
```

We provide a particular input for the demo.

```
initializeDigraph()
  let N = [new Vertex(i) | i in [0..9]]
  V := asSet(N)
  E := {(N(6),N(0)),(N(2),N(0)),(N(6),N(2)), (N(5),N(1)), (N(5),N(2)),
        (N(3),N(1)),(N(1),N(0)),(N(9),N(1)),(N(9),N(4)),(N(4),N(0)),
        (N(7),N(2)),(N(0),N(7))
}
```

This is the graphical form of the same.

```
C:\WINDOWS\system32\cmd.exe
Graph is cyclic.
Press any key to continue . . .
```

```
x where X(u,v))]
```

```
nteger field and a
```

```
let N = [new Vertex(i) | i in [0..9]]
V := set(N)
E := {(N(6),N(0)),(N(2),N(0)),(N(8),N(2)), (N(5),N(1)), (N(5),N(2)),
      (N(3),N(1)),(N(1),N(0)),(N(9),N(1)),(N(9),N(4)),(N(4),N(0)),
      (N(7),N(2)),(N(0),N(7))
}
```

This is the graphical form of the same.

Topsort

-
- (a more OO version)

```
class Vertex
```

This is an abstract class at this point - no fields or methods. We don't know what vertices are.

```
structure Digraph
  V as Set of Vertex
  E as Set of (Vertex,Vertex)
```

```
structure Digraph...
  topsort() as Seq of Vertex
    var S as Seq of Vertex = []
    step until fixpoint
      let U = V difference asSet(S)
      if U <> {} then
        S := S + [(any v | v in U where not(exists u in U where E(u,v)))]
    step
      return S
```

In order to print vertices, we endow Vertex with an integer field and a printing method.

```
class Vertex...
  key as Integer
  asString() as String
    return "v" + asString(key)
```

We provide a particular input for the demo.

```
mkDigraph() as Digraph
    let V = [new Vertex(k) | k in [0..10]]
```

Theory and tools behind our approach

- ASMs: abstract state machines
- AsmL: ASM Language
- Additional tools for testing and runtime verification

ASMs

- Maximal expressivity of algorithms
  - including distributed algorithms -
  on their natural abstraction levels:
  For every algorithm A, there is an ASM B that is simulates A step for step.
  - Experimental confirmation
  - Seq and parallel (non-distributed) versions have been proved.

AsmL, Microsoft's ASM-Language

A powerful specification language

- Combines mathematical, object-oriented and component-oriented approaches
- Massively parallel
  - Closer to query languages rather than PLs in that respect
- Integrated with VS6 and VS .Net
  - A member of the .Net language family: interoperable with other .Net languages
- Integrated with MS Word, XML

An example spec

- Trusted Windows

Page Table Edit Control: An AsmL Model

Foundations of Software Engineering
Microsoft Research

December 6, 2001

Abstract

This paper uses AsmL (the Abstract State Machine Language) to describe the behavior of a proposed security component called the Page Table Edit Control.

TW.doc - Microsoft Word

File Edit View Insert Format Tools Table Window Help

AsmL Word Turn off AsmL Word

Foundations of Software Engineering

Microsoft Research

December 6, 2001

Abstract

This paper uses AsmL (the Abstract State Machine Language) to describe the behavior of a proposed security component called the Page Table Edit Control

Section Break (Odd Page)

Page 1 Sec 1 1/92 At 1" Ln 1 Col 1 English (U.S

start

Foundations of Software Engineering

Microsoft Research

December 6, 2001

Abstract

This paper uses AsmL (the Abstract State Machine Language) to describe the behavior of a proposed security component called the Page Table Edit Control.

Section Break (Odd Page)

The asynchrony of buffered writes by multiple processors may be a security hole. An attack based on the asynchronous behavior of the system memory bus would proceed as follows:

- Figure out a way to manipulate the write buffer so that the writes of a particular processor drain into system memory as slowly as possible. For example, perform a large number of writes so that the processor's buffer is full.
- Select an unused page frame of system memory and format it as a page table. Choose some location *loc* in the new page as the target of attack. Write a word of memory into *loc* formatted as a page table entry whose target is a trusted page. Make sure that the operation goes into a "slow write buffer" from step one above.
- In parallel with the operation given in the previous bullet, add the page frame number of the page table from the previous step as a page table entry of some directory found in `System.RPV`. Do this in such a way that the page table edit will be resolved before the buffered write from the previous step has had a chance to be drained into system memory. This will require that two processors be used, one with a busy write buffer and one with a relatively idle one. (The attack cannot be performed with just one

Spec

Validate

Comprehend

Play scenarios

Test

Model check

Prove properties

Enforce

Generate test suites

On-the-fly testing

Lockstep runtime verification

Testing with AsmL

- We are deeply involved with testing.
  - Our approach empowers testers, and they like that.
- The ITE/AsmL project
- There are powerful testing techniques for finite automata
- An ASM-to-FSM algorithm

Spec

Validate

Comprehend

Play scenarios

Test

Model check

Prove properties

Enforce

Generate test suites

On-the-fly testing

Lockstep runtime verification